MMKV介绍

MMKV 是腾讯开源的一款基于 mmap 内存映射的 key-value 组件，底层序列化/反序列化使用 protobuf 实现，性能高，稳定性强，从 2015 年中至今在微信上使用，其性能和稳定性经过了时间的验证。
GitHub地址：https://github.com/Tencent/MMKV

为什么要替代SharedPreferences?

1，数据加密。 在 Android 环境里，数据加密是非常必须的，SP实际上是把键值对放到本地文件中进行存储。如果要保证数据安全需要自己加密，MMKV 使用了 AES CFB-128 算法来加密/解密。
2，多进程共享。系统自带的 SharedPreferences 对多进程的支持不好。现有基于 ContentProvider 封装的实现，虽然多进程是支持了，但是性能低下，经常导致 ANR。考虑到 mmap 共享内存本质上是多进程共享的，MMKV 在这个基础上，深入挖掘了 Android 系统的能力，提供了可能是业界最高效的多进程数据共享组件。
3，匿名内存。 在多进程共享的基础上，考虑到某些敏感数据(例如密码)需要进程间共享，但是不方便落地存储到文件上，直接用 mmap 不合适。而Android 系统提供了 Ashmem 匿名共享内存的能力，它在进程退出后就会消失，不会落地到文件上，非常适合这个场景。MMKV 基于此也提供了 Ashmem(匿名共享内存) MMKV 的功能。
4，效率更高。MMKV 使用protobuf进行序列化和反序列化，比起SP的xml存放方式，更加高效。
5，支持从 SP迁移，如果你之前项目里面都是使用SP，现在想改为使用MMKV，只需几行代码即可将之前的SP实现迁移到MMKV。

支持的数据类型

1，支持以下 Java 语言基础类型：

- boolean、int、long、float、double、byte[]

2，支持以下 Java 类和容器：

- String、Set< String >
- 任何实现了Parcelable的类型

使用

1，添加依赖

```
dependencies {
    compile 'com.tencent:mmkv:1.0.23'
}
123
```

mmkv从1.0.20版本开始迁移到AndroidX，具体版本历史请参看 CHANGELOG.md。

2，初始化
在自定义的Application中：

```
 MMKV.initialize(this);
1
```

3，数据操作（工具类封装）
MMKV 提供一个全局的实例，可以直接使用：

```
import com.tencent.mmkv.MMKV;
...
MMKV kv = MMKV.defaultMMKV();

// 存储数据很简单了，只需要调用如下一行代码即可，不用再如prefences一样调用apply或commit：
kv.encode("bool", true);
// 获取存储的信息
System.out.println("bool: " + kv.decodeBool("bool"));

kv.encode("int", Integer.MIN_VALUE);
System.out.println("int: " + kv.decodeInt("int"));

kv.encode("long", Long.MAX_VALUE);
System.out.println("long: " + kv.decodeLong("long"));

kv.encode("float", -3.14f);
System.out.println("float: " + kv.decodeFloat("float"));

kv.encode("double", Double.MIN_VALUE);
System.out.println("double: " + kv.decodeDouble("double"));

kv.encode("string", "Hello from mmkv");
System.out.println("string: " + kv.decodeString("string"));

byte[] bytes = {'m', 'm', 'k', 'v'};
kv.encode("bytes", bytes);
System.out.println("bytes: " + new String(kv.decodeBytes("bytes")));
12345678910111213141516171819202122232425262 7
```

4，删除 & 查询：

```
MMKV kv = MMKV.defaultMMKV();

kv.removeValueForKey("bool");
System.out.println("bool: " + kv.decodeBool("bool"));

kv.removeValuesForKeys(new String[]{"int", "long"});
System.out.println("allKeys: " + Arrays.toString(kv.allKeys()));

kv.clearAll();

boolean hasBool = kv.containsKey("bool");
1234567891011
```

5，如果不同业务需要区别存储，也可以单独创建自己的实例：

```
MMKV mmkv = MMKV.mmkvWithID("MyID");
mmkv.encode("bool", true);
12
```

6，默认是支持单进程的，如果业务需要多进程访问，那么在初始化的时候加上标志位 MMKV.MULTI_PROCESS_MODE：

```
MMKV mmkv = MMKV.mmkvWithID("InterProcessKV", MMKV.MULTI_PROCESS_MODE);
mmkv.encode("bool", true);
12
```

自定义根目录

MMKV 默认把文件存放在$(FilesDir)/mmkv/目录。你可以在 App 启动时自定义根目录：

```
String dir = getFilesDir().getAbsolutePath() + "/mmkv_2";
String rootDir = MMKV.initialize(dir);
Log.i("MMKV", "mmkv root: " + rootDir);
123
```

MMKV 甚至支持自定义某个文件的目录：

```
String relativePath = getFilesDir().getAbsolutePath() + "/mmkv_3";
MMKV kv = MMKV.mmkvWithID("testCustomDir", relativePath);
12
```

SharedPreferences 迁移

- MMKV 提供了 importFromSharedPreferences() 函数，可以比较方便地迁移数据过来。
- MMKV 还额外实现了一遍 SharedPreferences、SharedPreferences.Editor 这两个 interface，在迁移的时候只需两三行代码即可，其他 CRUD 操作代码都不用改。

```
private void testImportSharedPreferences() {
    //SharedPreferences preferences = getSharedPreferences("myData",
MODE_PRIVATE);
    MMKV preferences = MMKV.mmkvWithID("myData");
    // 迁移旧数据
    {
        SharedPreferences old_man = getSharedPreferences("myData",
MODE_PRIVATE);
        preferences.importFromSharedPreferences(old_man);
        old_man.edit().clear().commit();
    }
    // 跟以前用法一样
    SharedPreferences.Editor editor = preferences.edit();
    editor.putBoolean("bool", true);
    editor.putInt("int", Integer.MIN_VALUE);
    editor.putLong("long", Long.MAX_VALUE);
    editor.putFloat("float", -3.14f);
    editor.putString("string", "hello, imported");
    HashSet<String> set = new HashSet<String>();
    set.add("W"); set.add("e"); set.add("C"); set.add("h"); set.add("a");
set.add("t");
    editor.putStringSet("string-set", set);
    // 无需调用 commit()
    //editor.commit();
}
12345678910111213141516171819202122
```

工具类封装如下：

```
public class SpUtils {

    private static SpUtils mInstance;
    private static MMKV mv;

    private SpUtils2() {
        mv = MMKV.defaultMMKV();
    }

    /**
     * 初始化MMKV,只需要初始化一次，建议在Application中初始化
     *
     */
    public static SpUtils getInstance() {
        if (mInstance == null) {
            synchronized (SpUtils.class) {
                if (mInstance == null) {
                    mInstance = new SpUtils();
                }
            }
        }
        return mInstance;
    }

    /**
     * 保存数据的方法，我们需要拿到保存数据的具体类型，然后根据类型调用不同的保存方法
     *
     * @param key
     * @param object
     */
    public static void encode(String key, Object object) {
        if (object instanceof String) {
            mv.encode(key, (String) object);
        } else if (object instanceof Integer) {
            mv.encode(key, (Integer) object);
        } else if (object instanceof Boolean) {
            mv.encode(key, (Boolean) object);
        } else if (object instanceof Float) {
            mv.encode(key, (Float) object);
        } else if (object instanceof Long) {
            mv.encode(key, (Long) object);
        } else if (object instanceof Double) {
            mv.encode(key, (Double) object);
        } else if (object instanceof byte[] ) {
            mv.encode(key, (byte[]) object);
        } else {
            mv.encode(key, object.toString());
        }
    }

    public static void encodeSet(String key,Set<String> sets) {
        mv.encode(key, sets);
    }
```

```
    public static void encodeParcelable(String key,Parcelable obj) {
        mv.encode(key, obj);
    }


    /**
     * 得到保存数据的方法，我们根据默认值得到保存的数据的具体类型，然后调用相对于的方法获取值
     */
    public static Integer decodeInt(String key) {
        return mv.decodeInt(key, 0);
    }
    public static Double decodeDouble(String key) {
        return mv.decodeDouble(key, 0.00);
    }
    public static Long decodeLong(String key) {
        return mv.decodeLong(key, 0L);
    }
    public static Boolean decodeBoolean(String key) {
        return mv.decodeBool(key, false);
    }
    public static Float decodeFloat(String key) {
        return mv.decodeFloat(key, 0F);
    }
    public static byte[] decodeBytes(String key) {
        return mv.decodeBytes(key);
    }
    public static String decodeString(String key) {
        return mv.decodeString(key,"");
    }
    public static Set<String> decodeStringSet(String key) {
        return mv.decodeStringSet(key, Collections.<String>emptySet());
    }
    public static Parcelable decodeParcelable(String key) {
        return mv.decodeParcelable(key, null);
    }
    /**
     * 移除某个key对
     *
     * @param key
     */
    public static void removeKey(String key) {
        mv.removeValueForKey(key);
    }
    /**
     * 清除所有key
     */
    public static void clearAll() {
        mv.clearAll();
    }

}
```